link github: https://github.com/panty22/Progetto-2-Elezione-del-Coordinatore-con-Bully-Algorithm.git

Progetto 2: Elezione del Coordinatore con Bully Algorithm - Manuale Operativo

Questo elaborato illustra la procedura operativa per avviare il sistema distribuito basato sull'algoritmo del Bully (Bully Algorithm), con l'obiettivo di verificarne il funzionamento e la corretta gestione dei guasti.
L'architettura prevede l'esecuzione simultanea di 5 nodi indipendenti, ciascuno gestito tramite una finestra di terminale dedicata.

1. Apertura dei Terminali

Avviare 5 finestre distinte di PowerShell (o Prompt dei Comandi). Ogni finestra rappresenterà un nodo del sistema.

Posizionamento nella Cartella di Lavoro
In ciascun terminale, digitare il seguente comando per accedere alla directory del progetto: `cd "percorso\cartella\progetto"`

(Esempio: `cd "C:\Users\nome\Desktop\BullyGithub"`)

Nota: È possibile recuperare facilmente il percorso corretto facendo clic destro sul file `Node.java` > *Proprietà* > copiare il percorso indicato in "Percorso". Questo passaggio va eseguito una sola volta per ogni terminale aperto.

Compilazione del Codice Sorgente
In uno qualsiasi dei terminali (è sufficiente farlo una volta sola), digitare il comando: `javac Node.java`

Questo comando compila il codice sorgente, trasformando il file di testo leggibile (`Node.java`) in bytecode eseguibile (`Node.class`) che potrà essere interpretato dalla Java Virtual Machine.

Avvio dei Nodi
Avviare i nodi in sequenza, uno per ogni terminale, assegnando a ciascuno un ID univoco progressivo. Si consiglia di attendere un paio di secondi tra un comando e l'altro per permettere la corretta inizializzazione:

- Terminale 1: `java Node 1`

- Terminale 2: `java Node 2`
- Terminale 3: `java Node 3`
- Terminale 4: `java Node 4`
- Terminale 5: `java Node 5`

Una volta avviati tutti i nodi, il sistema avvierà automaticamente l'elezione del coordinatore (P5). Per testare la gestione dei guasti, sarà sufficiente terminare il processo del nodo coordinatore (premendo `Ctrl+C` nel suo terminale) e osservare la nuova elezione automatica.

1.1 Linguaggio e Ambiente

- Linguaggio di programmazione: Java
- Versione del linguaggio: Java SE 25 (Versione "25.0.2" - 2026-01-20 LTS)
- Sistema operativo di riferimento: Microsoft Windows 10

1.2 Librerie e Dipendenze

Il progetto è sviluppato utilizzando esclusivamente le librerie standard incluse nel Java Development Kit (JDK). Non sono richieste librerie di terze parti o dipendenze esterne.
Le librerie principali utilizzate sono quelle standard della piattaforma Java SE; sono incluse di default in qualsiasi installazione di JDK. Non sono stati applicati comandi per l'installazione delle dipendenze perché non sono state utilizzate librerie esterne.

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.*;
```

1.3 Compilazione

Il progetto è costituito da un unico file sorgente Java e non richiede strumenti di build complessi.

Comando di compilazione:
Aprire il terminale nella directory principale del progetto ed eseguire il seguente comando: `javac Node.java`

Se la compilazione ha successo, il comando non restituisce output e viene generato il file bytecode `Node.class` nella stessa cartella.

1.4 Avvio dei Nodi

Per eseguire il sistema distribuito, è necessario aprire 5 finestre di terminale separate (si consiglia l'uso di PowerShell, ma è possibile utilizzare anche il Prompt dei Comandi).

In ciascun terminale, posizionarsi innanzitutto nella directory del progetto digitando il comando: `cd "C:\Users\nome\Desktop\BullyGithub"`

Successivamente, avviare i nodi in sequenza digitando i seguenti comandi nelle rispettive finestre:

- Terminale 1: `java Node 1` (avvia il nodo P1 sulla porta 5001)
- Terminale 2: `java Node 2` (avvia il nodo P2 sulla porta 5002)
- Terminale 3: `java Node 3` (avvia il nodo P3 sulla porta 5003)
- Terminale 4: `java Node 4` (avvia il nodo P4 sulla porta 5004)
- Terminale 5: `java Node 5` (avvia il nodo P5 sulla porta 5005)

Il sistema utilizza l'indirizzo `localhost` per la comunicazione tra i nodi, ciascuno dei quali si pone in ascolto sulla porta TCP dedicata.
Si raccomanda di avviare i nodi in ordine crescente (da P1 a P5), attendendo qualche secondo tra un avvio e l'altro per garantire la corretta inizializzazione dei socket di rete. Ogni nodo confermerà l'avvenuta attivazione stampando a video il messaggio: *"Server attivo su porta 500X"*.

Una volta che tutti i nodi saranno attivi, il sistema avvierà automaticamente la procedura di elezione.

1.5 Associazione Nodo–Terminale

L'associazione tra nodo e terminale avviene in modo manuale all'avvio del programma ed è gestita nel metodo `main` del codice. Ogni terminale esegue un'istanza separata della classe `Node`, passando l'identificativo univoco (ID) come argomento da riga di comando.

Nel metodo `main`, l'argomento `args[0]` viene letto e convertito in intero per definire l'identità del nodo corrente all'interno di quella specifica sessione di terminale.

Quando l'utente digita `java Node 1` nel terminale:

1. Il terminale avvia la JVM.
2. La JVM passa la stringa `"1"` all'array `args`.
3. Il codice la converte in `int processId = 1`.
4. Quella specifica finestra di terminale diventa "vincolata" all'esecuzione del nodo P1.

```java
// Punto di ingresso del processo: verifica argomenti, inizializzazione nodo
    Run | Debug
    public static void main(String[] args) {
        if (args.length < 1) {// Verifica presenza argomento ID processo
            System.err.println(x: "Errore: specificare l'ID del processo");//
            System.err.println(x: "Sintassi: java Node <id>");
            System.exit(status: 1);
        }// Parsing ID processo e validazione intervallo
// In caso di errore, stampa messaggio e termina processo
        try {
            int processId = Integer.parseInt(args[0]);
            if (processId < 1 || processId > TOTAL_PROCESSES) {
                System.err.println("ID processo deve essere tra 1 e " + TOTAL_PROCESSES);
                System.exit(status: 1);
            }
// Creazione e inizializzazione nodo
            Node node = new Node(processId);
            node.initialize();
        } catch (NumberFormatException e) {
            System.err.println("ID processo non valido: " + args[0]);
            System.exit(status: 1);
        } catch (Exception e) {
            System.err.println("Errore: " + e.getMessage());
            System.exit(status: 1);
        }
    }
```

Riconoscimento del Nodo dai Log

È possibile riconoscere a colpo d'occhio quale nodo è in esecuzione in uno specifico terminale leggendo il prefisso presente in ogni riga di output.
Il sistema di logging antepone automaticamente l'etichetta `[Px]` a ogni messaggio stampato, dove `x` corrisponde all'ID del nodo associato a quella finestra.

Esempi:

- `[P1] Server attivo su porta 5001` → Terminale del NODO 1
- `[P5] Mi dichiaro COORDINATORE` → Terminale del NODO 5

Questa convenzione visiva permette di monitorare il comportamento distribuito in tempo reale senza dover ricordare manualmente quale comando è stato lanciato in ciascuna finestra.

```java
// Utility per log: formato uniforme con identificativo processo, utile per debug e tracciamento flusso esecuzione
    private void log(String message) {
        System.out.println("[P" + processId + "] " + message);
    }//log con identificativo processo per tracciamento chiaro delle operazioni e stato del nodo
}// Fine implementazione algoritmo Bully per elezione del coordinatore in ambiente distribuito
```